实习总结

李伟豪

北京星天地信息科技有限公司

2024/05/28 - 2024/07/31

目录

第一章　实习目标

1.1　学习方向及目标

学习现在先进的计算机视觉（Computer Vision）以及图形学与 AI 结合的模型如 Neural Radiance Fields（神经辐射场，简称 NeRF），3D Gaussian Splatting（3D 高斯溅射，简称 3dgs），以及 YOLO（全称 You Only Look Once），同时理解各个模型的实现原理。

1.2　期望

- 完成搭建尽可能多的 AI 模型，配置其环境，并完成训练其预设数据库。
- 学习并理解各个模型的实现原理。
- 自己制作数据并通过基于 AI 的三维重建制作模型。

第二章　模型的搭建以及学习

2.1　Yolo V5

2.1.1　模型简介

Yolo(You Only Look Once) 是一种单阶段目标检测算法，即仅需要“看”一次就可以识别出图片中物体的 class 类别和边界框。Yolov5 是由 Alexey Bochkovskiy 等人在 YOLO 系列算法的基础上进行改进和优化而开发的，使其性能与精度都得到了极大的提升。

2.1.2　模型环境配置

- Python = 3.8.19
- torch = 2.3
- torchvision = 0.18.0
- gitpython = 2.40.1
- opencv-python = 4.9.0.80
- matplotlib = 3.7.5
- pandas = 2.0.3

模型从https://github.com/ultralytics/yolov5.git 克隆下来，然后在本地进行配置。

2.1.3　模型训练

模型训练是在train.py 文件中进行的，训练时可以同时输入 data[1]、cfg[2]和 weight[3]文件；或是 epochs[4]和 batch size[5]。

如下图：

[1]文件中包含了训练集和验证集的路径，以及所有的标注种类

[2]文件中包含了模型的参数，如学习率、batch size 等

[3]文件中包含了预训练模型的权重

[4]训练的次数

[5]每次训练的图片数量

```
python train.py --data coco.yaml --epochs 300 --weights '' --cfg yolov5n.yaml  --batch-size 128
                                                          yolov5s                    64
                                                          yolov5m                    40
                                                          yolov5l                    24
                                                          yolov5x                    16
```

图 2-1 YOLOv5 目标检测模型的训练命令

2.1.4 测试自己制作的数据集

从网上找寻了 30 张宝可梦的图片，然后通过 labelImg 工具[6]标注这些图片中比较常见的宝可梦，如皮卡丘、杰尼龟等，然后将这些图片和标注文件放入到一个文件夹中。最后通过 YOLOv5 模型进行训练，得到了一个可以识别这些宝可梦的模型，因为数据集比较小，所以模型的识别率不是很高。

训练结果示例：

图 2-2 YOLOv5 目标检测模型的训练结果

2.1.5 实现原理

YOLOv5 首先会在输入端中将输入图片进行预处理，图像大小调整为模型所需的大小，进行归一化操作，及将像素值缩放到 0 到 1 之间。

然后将图片输入到 backbone 网络[7]中，backbone 网络会将图片的特征提取出来，然后将这些特征输入到 neck 网络[8]中，neck 网络会将不同层次的特征进行融合，然后将这些特征输入到 head 网络[9]中，head 网络会将图片中的物体进行预测，

[6]LabelImg 是一个开源的图像标注工具，用于创建图像数据集时标记图像中的对象边界框

[7]backbone 网络是一个特征提取网络，用于提取图片的特征

[8]neck 网络是一个特征融合网络，用于将不同层次的特征进行融合

[9]head 网络是一个预测网络，用于预测图片中的物体

得到物体的类别和边界框。

下图为 YOLOv5 的网络结构：[10] [11] [12] [13] [14] [15] [16]

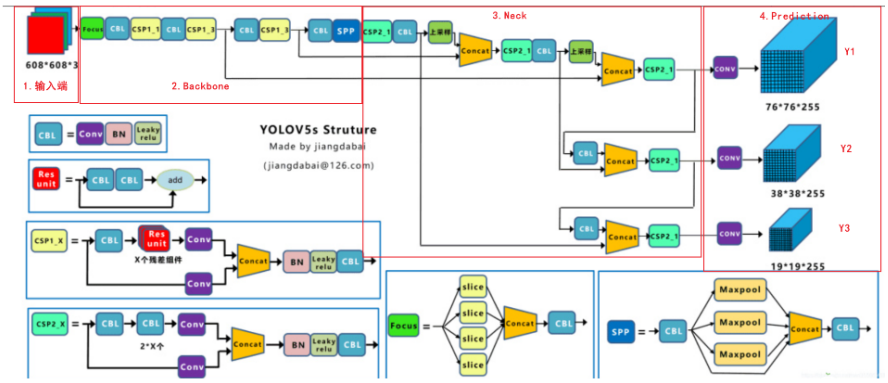

图 2-3 YOLOv5 的网络结构

[10]CPL：由卷积 Conv+ 批量归一化 BN+ 激活函数 Leaky Relu 组成，用于在特征提取的过程中增加网络的信息传递能力。CPL 通过在不同阶段的特征图之间进行部分连接，使得网络可以更充分地利用低级和高级特征，从而提高模型的性能。

[11]CSP1：由 CBL 模块、Res uint 模块以及卷积层 Concat 组成。CSP1 是 CPL 的一种变体，它在 CPL 的基础上引入了跨阶段的残差连接，以进一步增强网络的信息传递能力。CSP1 可以有效地减少网络中的计算量，提高模型的效率。

[12]CSP2：不再使用 Res unit，由卷积层 CBL 模块 Concat 组成。CSP2 引入了路径重排的技术，以重新排列网络中的路径，使得前向和后向传播之间的信息流更加平滑和高效。这种路径重排有助于减少网络中的计算量，提高模型的速度和效率。CSP2 同样也进行了路径融合（Path Fusion），即在不同路径之间进行信息融合，以使网络可以更充分地利用低级和高级特征。这种路径融合有助于提高模型对目标的检测和识别能力。CSP1 也具有类似的路径融合机制。

[13]SPP 是一种用于空间金字塔池化的技术，它能够在不同尺度下提取图像的局部特征。SPP 在 YOLOv5 中用于提取特征图的空间信息，从而使得模型可以更好地检测不同尺度和大小的目标。

[14]Focus: 对图像进行切片后再 Concat

[15]上采样：利用元素复制扩充的方法使得特征图尺寸扩大，例如线性插值。

[16]Concat：张量拼接，会扩充两个张量的维度，实现多尺度特征融合。

2.2 NeRF

2.2.1 模型简介

NeRF（Neural Radiance Fields）是一种用于三维重建的深度学习模型，它可以从二维图像中重建出高质量的三维场景。NeRF 的核心思想是利用神经网络来建模场景中每个空间点的辐射亮度和密度，从而实现对场景的准确重建。

2.2.2 模型环境配置

于 CUDA 10.2 基础下：

- Python = 3.7.1
- cudatoolkit = 10.0.130
- tensorflow = 1.14.0
- numpy = 1.21.5
- imageio = 2.193
- configargparse = 1.4

模型从https://github.com/bmild/nerf.git 克隆下来，然后在本地进行配置。

2.2.3 模型训练

模型训练是在run_nerf.py 文件中进行的，需要同时输入配置文件的的相对路径，配置文件中包含了实验名称（expname），基础目录（basedir），数据集路径（datadir），数据集类型（dataset_type）采样参数、使用视角方向信息等模型训练和数据配置信息。

2.2.4 训练结果

先在 730 显卡的电脑上配置环境并尝试训练，但因预计训练结束时间要 20 左右，后改为在 3080 显卡的电脑上重新配置环境。在重新配置环境时遇到一些版本冲突的问题，因为 3080 显卡的电脑上的 CUDA 版本是 12.3，不能与原配置的 tensorflow 版本 1.14.0 兼容，所以需要重新配置环境。最后在 3080 显卡的电脑上重新配置环境并训练，但最后的预计训练结束时间人在一天左右，所以在 github 上找到了另外一个模型 NeRF Pytorch（详情见2.3 NeRF Pytorch）。

2.2.5 实现原理

简单一点理解可以把 NeRF 看作是先用深度全连接神经网络（deep fully-connected neural network）[17]去学习并训练一个静态 3D 场景，实现复杂场景的任意新视角合成（渲染）。NeRF 的核心思想是将静态场景表示为 5D 函数，即每个空间点 x(x, y, z) 和方向 d$(\theta$[18]$, \phi$[19]$)$ 都对应一个颜色（RGB）和密度 (σ)。NeRF 的输入是一条被 5D 函数代表的射线，输出是射线上每个点的颜色和密度。其中方向 d(θ, ϕ)，在进入模型前将先会被展开成三维笛卡尔坐标 (3D Cartesian) 系下的单位向量 $\vec{u}$。这种单位向量可以用来表示一个方向, 它有着确定的 x, y, z 三个分量, 记为 (ux, uy, uz)。由于是单位向量, 所以这三个分量要满足 (公式 2.1)：

$$u_x^2 + u_y^2 + u_z^2 = 1 \tag{2-1}$$

(ux, uy, uz) 具体的计算公式如下（公式 2-2）：

$$\begin{aligned} u_x &= \sin(\theta)\cos(\phi) \\ u_y &= \sin(\theta)\sin(\phi) \\ u_z &= \cos(\theta) \end{aligned} \tag{2-2}$$

在进入模型前, 通常还会先将 (x, y, z) 和 (ux, uy, uz) 通过位置编码 (positional encoding)[20]映射到更高维的周期性函数空间[21]中, 以增加模型的表达能力。转换公式示例如下（公式 2-3）：

$$r(x) = (\sin(2^0\pi x), \cos(2^0\pi x), \sin(2^1\pi x), \cos(2^1\pi x), \cdots, \sin(2^{L-1}\pi x), \cos(2^{L-1}\pi x)) \tag{2-3}$$

及将一个一维向量 x 映射到一个 $2L$ 维的向量，在论文中作者对空间点 $x(x, y, z)$ 转换成了 60 维的向量，即 $L = 10$，$3 \times 2L(L = 10) = 60$，同时将方向 d

[17]通常被称为多层感知机（multilayer perceptron）或 MLP，没有卷积层（convolutional layers），由全连接层（fully connected layer）和激活函数（activation function）组成

[18]极角（Polar Angle）, 通常的取值范围是 0 到 π 之间，通常表示从参考方向（通常是参考坐标系的正方向）到目标向量（或目标位置）的角度，简单一点可以理解为与 z 轴的夹角

[19]方位角（Azimuthal Angle）, 通常的取值范围是 0 到 2π 之间，通常表示目标向量在参考平面上投影与参考方向的夹角，简单一点可以理解为在 xy 平面上与 x 轴的夹角

[20]位置编码是一种将空间位置信息映射到更高维的空间中的技术, 它可以帮助模型更好地理解空间位置信息, 从而提高模型的性能。

[21]周期性函数空间是指由周期函数构成的函数空间。一个周期性函数 $f(x)$ 是指满足 $f(x+T) = f(x)$ 的函数, 其中 T 是一个常数, 称为周期。

从 3 维 (ux, uy, uz) 转换成了 24 维的向量，即 $L = 4$，$3 \times 2L(L = 4) = 24$。（详情见图 2-5）。下图为论文实验时使用位置编码和不使用位置编码的对比图（图 2-4）：

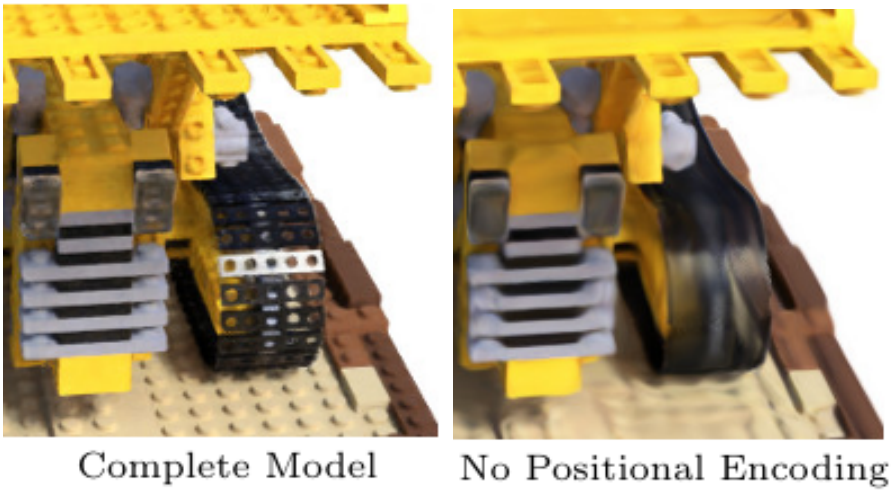

图 2-4 使用位置编码和不使用位置编码的对比图

从上图中可以看出在使用了位置编码后，模型在细节的表达中会更好，尤其是在凹凸不平的地方，而不使用位置编码的模型在这些地方会变得模糊。

在实际使用位置编码时除了上述公式中的 $2L$ 维向量外，有时还会加上一个维度为 1 的向量和表示其本身的向量进行拼接成为 $2L + 2$ 维的向量（论文的实验中并未使用 $2L + 2$ 维的向量，这里仅仅只是补充说明），这两个向量的加入可以帮助模型更好的学习低频平滑部分，同时提高模型的灵活性。加入后的新向量如下（公式 2-4）：

$$r(x) = (\sin(2^0\pi x), \cos(2^0\pi x), \sin(2^1\pi x), \cos(2^1\pi x), \cdots, \sin(2^{L-1}\pi x), \cos(2^{L-1}\pi x), x, 1) \tag{2-4}$$

空间点 $x(x, y, z)$ 通过位置编码升维过后的高维向量会先进入深度全连接神经网络（MLP）中，在经过多层的全连接层和激活函数 (Activation Function) 后，最后输出密度（σ）和一个维度和之前全连接层维度相同的中间特征向量，中间特征会和三维方向向量 d 通过位置编码升维过后的高维向量一起输入到额外的全连接层中去预测颜色（RGB）。示例如下图（图 2-5）：

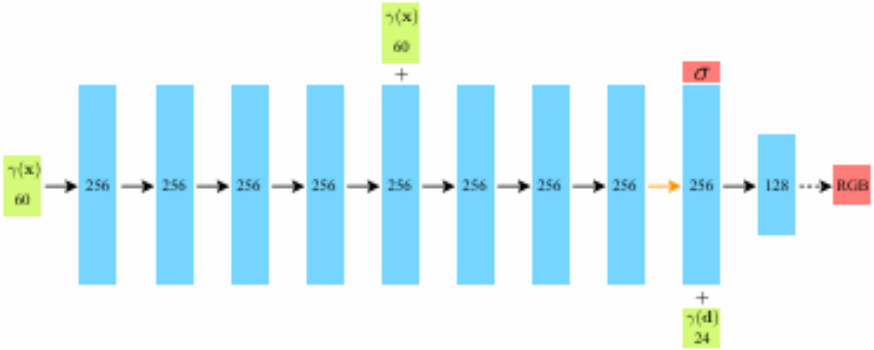

图 2-5 NeRF 中的 MLP 结构示例

2.3 NeRF Pytorch

致谢

致 谢

致谢内容。

文献

文 献

Mildenhall, B., Srinivasan, P. P., Tancik, M., Barron, J. T., Ramamoorthi, R., & Ng, R. (2020). NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis. In European Conference on Computer Vision (ECCV).